Boosting Interval-Based Time Series Classifiers

Carlos J. Alonso González, Juan J. Rodríguez Diez

Abstract—**A supervised classification method for time series, even multivariable, is proposed. It is based on boosting very simple classifiers. They are formed only by one literal. The used predicates, such as "always" and "sometime" operate over temporal intervals and regions in the dominion of the values of the variable. These regions are obtained previously, using discretization techniques.**

The experimental validation of the method has been done using different data sets, some of them obtained from the UCI repositories. The results are very competitive with the obtained in previous works. Moreover, their comprehensibility is better than in other approaches with similar results, since the classifiers are formed by a weighted sequence of literals.

Keywords—**Time Series, Boosting, Machine Learning**

I. Introduction

Multivariate time series classification is useful in domains such as biomedical signals [1], continuous systems diagnosis [2] and data mining in temporal databases [3]. This problem can be tackled extracting features of the series, through some kind of preprocessing, and using some conventional machine learning method. Nevertheless, this approach has several drawbacks, these techniques are usually *ad hoc* and domain specific [4]. The design of specific machine learning methods for the induction of time series classifiers allows the construction of more comprehensible classifiers in a more efficient way.

We propose a simple, although effective, technique for time series classification based on literals over temporal intervals (e.g., always(Variable, Region, Beginning, End)) and boosting (a method for the generation of ensembles of classifiers) [5].

The rest of the paper is organised as follows. The base classifiers are described in section II. Boosting these classifiers is explained in section III. Section IV presents the experimental validation. Finally, section V concludes.

II. Base Classifiers

A. Temporal Predicates

The selection and definition of the temporal predicates is based in the ones used in a visual rule language for dynamic systems [2] and it is introduced in [6]. The temporal predicates used are the following:

- always(Variable, Region, Beginning, End). It is true if the Variable is always in this Region in the interval between Beginning and End.
- sometime(Variable, Region, Beginning, End). It is true if the Variable is sometime in this Region in the interval between Beginning and End.
- true_precentage(Variable, Region, Beginning, End, Percentage). It is true if the percentage of the time between Beginning and End where the variable is in Region is greater or equal than Percentage.

Once that it is decided to work with temporal intervals, the use and definition of the predicates always and sometime is natural, due to the fact that they are the extension of the conjunction and disjunction to intervals. Since one appears too demanding and the other too flexible, a third one has been introduced, true_precentage. It is a "relaxed always" (or a "restricted sometime"). The additional parameter indicates the degree of flexibility (or restriction).

A.1 Regions.

The regions that appear in the previous predicates are intervals in the domain of values of the variable. In some cases the definitions of these regions can be obtained from an expert, as background knowledge. Otherwise, they can be obtained with a discretization preprocess, which obtains r disjoint, consecutive intervals. The regions considered are these r intervals (equality tests) and others formed by the union of the intervals $1 \dots i$ (less or equal tests).

The reasons for fixing the regions before the classifier induction, instead of obtaining them while inducing, are efficiency and comprehensibility. The literals are easier to understand if the regions are few, fixed and not overlapped.

B. Searching Literals

Given a set of positive and negative examples, the best literal, according to some criterion, must be selected. Then, it is necessary to search over the space of literals. The possible number of intervals, if each series has n points, is $(n^2 - n)/2$. If p is the number of predicates considered, and v the total number of regions in the different variables, the possible number of atoms is $pv(n^2-n)/2$, and the possible number of literals (atoms possibly negated) is $pv(n^2 - n)$. In the case of predicates with additional arguments (true_precentage), it is also necessary to consider how many values are possible for them.

B.1 Linear Probing.

The process starts with windows of size 1 (between two consecutive points) and the windows of size $i + 1$ are evaluated from the windows of size i with the same origin. The initialisation of the windows of size 1, for each predicate, it is done in $O(e)$, where e is the total number of examples. The amplification of the size of the window in one unit is also done in $O(e)$. It is necessary to calculate the number of positive and negative examples covered. In the most complex case, true_precentage, this

This work has been supported by the Spanish CYCIT project TAP 99–0344.

Carlos J. Alonso González and Juan J. Rodríguez Diez are with the *Departamento de Informática, Universidad de Valladolid*, Spain. E-mail: calonso@infor.uva.es, juanjo@infor.uva.es .

requires a time of $O(e+f)$, where f is the number of values allowed in the additional argument of this predicate. Summarising, the worst execution time for finding the best literal is $O((e+f)pvn^2)$. If $e > f$ (the most common case) or true_precentage is not used, then it is $O(epvn^2)$.

B.2 Exponential Probing.

With the objective of reducing the search space, another alternative is considered. In this case not all the windows are probed. Only those that are power of 2 are considered. The number of these windows is of $\sum_{i=1}^{k}(n-2^{i-1}) = kn - 2^k - 1$ where $k = \lfloor \lg n \rfloor$. Using a dynamic programming algorithm it is possible to obtain the information necessary of the window of size $2i$ from two consecutive windows of size i, with a time of $O(e)$ (in the case of true_precentage, $O(e+f)$). The selection of the best literal in this case requires a time of $O(epvn \lg n)$.

III. Boosting

Nowadays, an active research topic is the use of *ensembles* of classifiers. They are obtained by generating and combining base classifiers, constructed using other machine learning methods, typically decision trees. The target of these ensembles is to increase the accuracy with respect to the base classifiers.

One of the most popular methods for creating ensembles is boosting [5], a family of methods, of which AdaBoost is the most prominent member. They work assigning a weight to each example. Initially, all the examples have the same weight. In each iteration a base classifier is constructed, according to the distribution of weights. Afterwards, the weights are readjusted according to the result of the example in the base classifier. The final result is obtained combining the weighted votes of the base classifiers.

Following the good results of works using ensembles of very simple classifiers [7], sometimes named *stumps*, in our case the base classifiers are formed by only one literal. Table I shows one of this classifier. The reasons for using so simple base classifiers are:

- Ease of implementation. In fact, it is simpler to implement a boosting algorithm than a decision tree or rule inducer. A first approximation to the induction of rules with these literals is described in [6].
- Comprehensibility. It is easier to understand a sequence of weighted literals than a sequence of weighted decision trees or rules.

The criterion used for selecting the best literal is to select the one with less error, relative to the weights, although preferring the literals with bigger intervals.

A. Multiclass problems

The simplest AdaBoost algorithm is defined for binary classifications problems [5], although there are extensions for multiclass problems [8]. In our case the base classifiers are also binary (only one literal) and it excludes some techniques for handling multiclass problems. We have used a simple approximation: the problem is reduced to several binary classification problems, as many as classes, which decides if an example is, or is not, of the corresponding class. Every binary problem is solved independently using boosting.

The advantages of using boosting for binary problems are that it is always possible to find a literal with an error less than 0.5, necessary for the boosting algorithm, since the problem is binary, and that the results are more comprehensible because they are organised by classes.

To classify a new example, it is evaluated by all the binary classifiers. If only one of them classifies it as positive, then the example is assigned to the corresponding class. If the situation is not so idyllic, we can consider that the multiclass classifier is not able to handle this example. This is a very pessimistic attitude. In the experiments, the classification error obtained with this point of view is named *maximum error*.

When using boosting in a binary problem, the result is positive or negative depending of the sign of the sum of the results of the individual classifiers, conveniently weighted. In a multiclass problem, if we have conflicts between several of the binary classifiers we use these sums of weights, normalised to $[-1, 1]$, to select the winner. In the experiments, the error obtained with this method is called *combined error*.

TABLE II

Characteristics of the data sets

Data set	Classes	Examples	Points
CBF	3	798	128
Control charts	6	600	60
Waveform	3	900	21
Wave + noise	3	900	40

IV. Experimental Validation

A. Data sets

The characteristics of the data sets are summarised in table II. The main criterion for selecting them was that the number of examples available were big enough, to ensure that the results were reliable. Figure 1 shows some examples.

A.1 Cylinder, Bell and Funnel (CBF).

This is an artificial problem, introduced by Saito [9]. The learning task is to distinguish between these three classes: cylinder (c), bell (b) or funnel (f). Examples are generated using the following functions:

$$\begin{aligned} c(t) &= (6+\eta)\cdot\chi_{[a,b]}(t) + \epsilon(t) \\ b(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(t-a)/(b-a) + \epsilon(t) \\ f(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(b-t)/(b-a) + \epsilon(t) \end{aligned}$$

where

$$\chi_{[a,b]} = \begin{cases} 0 & \text{if} \quad t < a \vee t > b \\ 1 & \text{if} \quad a \leq t \leq b \end{cases}$$

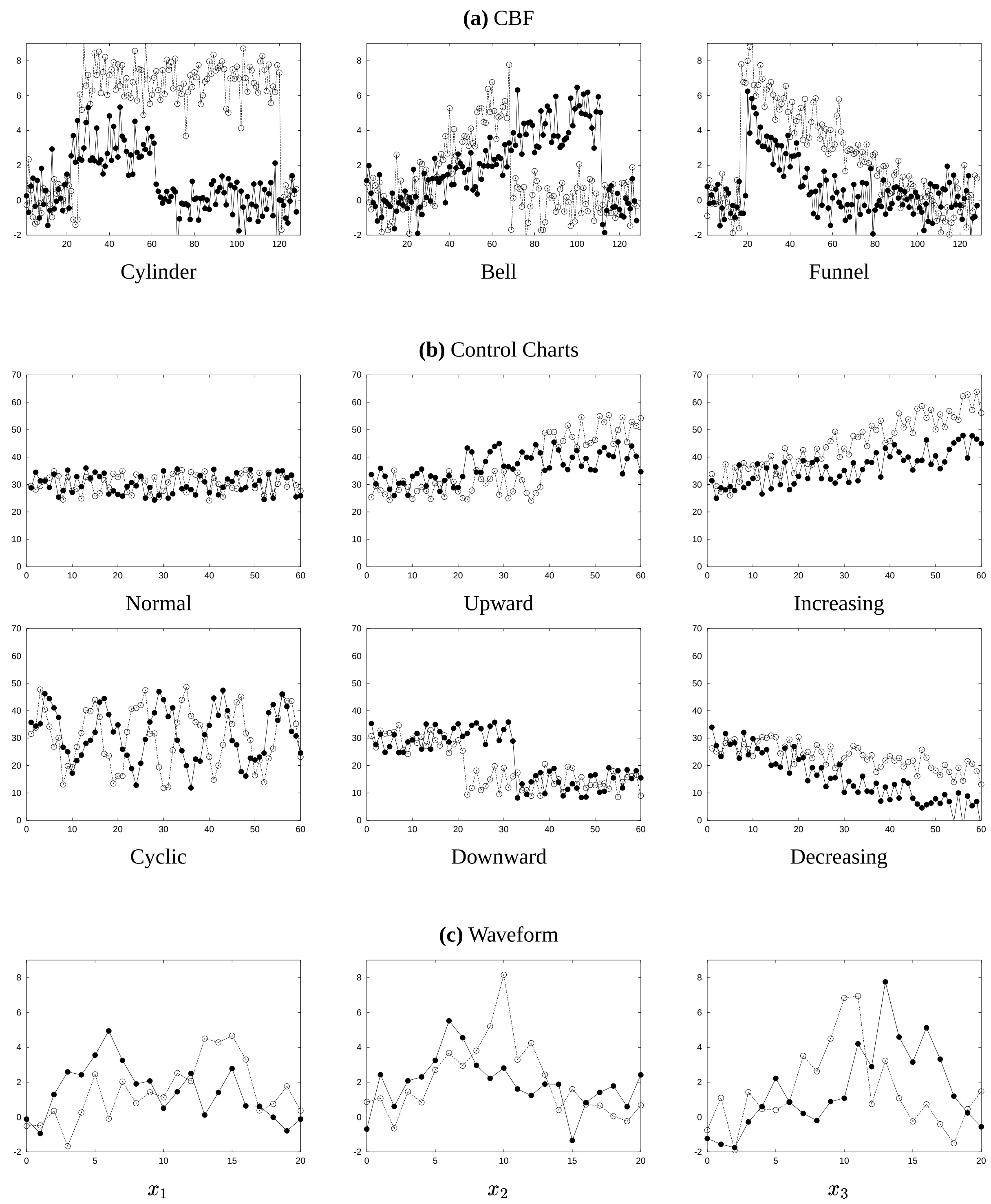

Fig. 1. Some examples of the data sets. Two examples of the same class are shown in each graph.

TABLE I

EXAMPLE OF CLASSIFIER. IT CORRESPONDS TO THE CLASS CYLINDER OF THE CBF DATA SET. IT SHOWS THE LITERAL, THE NUMBER AND TOTAL WEIGHT OF THE COVERED EXAMPLES AND THE ERROR AND WEIGHT OF THE INDIVIDUAL CLASSIFIER. THE NUMBER OF POSITIVE AND NEGATIVE EXAMPLES IN THE TRAINING SET ARE 213 AND 426, RESPECTIVELY.

literal	examples		weights		literal	
	$\oplus$	$\ominus$	$\oplus$	$\ominus$	error	weight
not true_precentage(x, 1_4, 20, 84, 50)	192	18	0.30	0.03	0.06	1.37
not true_precentage(x, 3, 37, 101, 15)	194	53	0.34	0.06	0.14	0.89
true_precentage(x, 5, 17, 33, 15)	135	134	0.45	0.09	0.14	0.91
not true_precentage(x, 4, 23, 87, 15)	170	10	0.28	0.01	0.17	0.78
not true_precentage(x, 3, 26, 90, 15)	207	46	0.56	0.09	0.17	0.79
not true_precentage(x, 4, 72, 74, 5)	194	225	0.55	0.15	0.18	0.77
true_precentage(x, 4, 20, 24, 30)	15	10	0.22	0.01	0.21	0.67
not true_precentage(x, 1_4, 45, 53, 85)	213	170	0.62	0.19	0.19	0.72
not true_precentage(x, 3, 24, 88, 5)	159	0	0.15	0.00	0.24	0.58
true_precentage(x, 5, 4, 36, 5)	178	190	0.57	0.19	0.22	0.63

and η and $\epsilon(t)$ are obtained from a standard normal distribution $N(0,1)$, a is an integer obtained from a uniform distribution in $[16,32]$ and $b-a$ is another integer obtained from another uniform distribution in $[32,96]$. The examples are generated evaluating those functions in $1,2\ldots 128$. For ease of comparison with previous results, 266 examples of each class were generated. Figure 1.a shows some examples of this data set.

A.2 Control Charts.

In this data set there are six different classes of control charts, synthetically generated by the process in [10]. Each time series is of length n, and is defined by $y(t)$, with $1 \leq t \leq n$:

1. Normal: $y(t) = m + rs$. Where $m = 30$, $s = 2$ and r is a random number in $[-3,3]$.
2. Cyclic: $y(t) = m + rs + a\sin(2\pi t/T)$. a and T are in $[10,15]$.
3. Increasing: $y(t) = m + rs + gt$. g is in $[0.2, 0.5]$.
4. Decreasing: $y(t) = m + rs - gt$.
5. Upward: $y(t) = m + rs + kx$. x is in $[7.5, 20]$ and $k = 0$ before time t_3 and 1 after this time. t_3 is in $[n/3, 2n/3]$.
6. Downward: $y(t) = m + rs - kx$.

The data used was obtained from the UCI KDD Archive [11]. It contains 100 examples of each class, with 60 points in each example. Figure 1.b shows some examples of this data set.

A.3 Waveform.

This data set was introduced by [12]. The purpouse is to distinguish between three classes, defined by the evaluation in $1,2\ldots 21$, of the following functions:

$$\begin{aligned} x_1(i) &= uh_1(i) + (1-u)h_2(i) + \epsilon(i) \\ x_2(i) &= uh_1(i) + (1-u)h_3(i) + \epsilon(i) \\ x_3(i) &= uh_2(i) + (1-u)h_3(i) + \epsilon(i) \end{aligned}$$

where $h_1(i) = \max(6 - |i-7|, 0)$, $h_2(i) = h_1(i-8)$, $h_3(i) = h_1(i-4)$, u is a uniform aleatory variable in $(0,1)$ and $\epsilon(t)$ follows a standard normal distribution.

We use the version from the UCI ML Repository [13]. Figure 1.c shows some examples of this data set.

A.4 Wave + Noise.

This data set is generated in the same way than the previous one, but 19 points are added at the end of each example, with mean 0 and variance 1. Again, we used the first 300 examples of each class of the corresponding data set from the UCI ML Repository.

B. Results

The results for each data set were obtained using five five-fold stratified cross-validation. The number of regions for each variable was 6. Only, the predicate true_precentage was used. The allowed values for the parameter percentage where $5, 15, 30, 50, 70, 85$ and 95. Boosting was applied with 50 iterations. Table III and figure 2 resume the results.

Globally, we can highlight the good evolution of the maximum error for each data set with the number of iterations in the boosting process. For all of the data sets, the best results appear in the last iteration, and the evolution is, nearly always, decreasing. For the combined error, the evolution is not so good. The last iteration is the best result only in one case (CBF), and in this case this result appears also in a previous iteration.

B.1 Cylinder, bell and funnel.

The best result, to our knowledge, previously published, with this data set is an error of 1.9 [4], using 10 fold cross validation. From the iteration 10, the results shown in table III are better than this result.

B.2 Control charts.

The best result is obtained in the iteration 30, with an error of 0.87. From the iteration 20, all the values are less or equal than 1.10. The only results we know with this data set are for similarity queries [10], and not for supervised classification.

TABLE III
EXPERIMENTAL RESULTS. FOR EACH DATA SET, THE FIRST COLUMN SHOWS, IN PERCENTAGE, THE MAXIMUM ERROR AND THE SECOND ONE THE COMBINED ERROR. IN BOLDFACE, THE BEST RESULTS OF EACH COLUMN.

Iter.	CBF		Control		Wave		Wave + Noise	
1	13.86	11.13	33.70	23.53	40.78	26.40	42.82	29.76
5	5.97	2.46	11.13	2.00	32.27	17.38	30.93	18.73
10	4.56	1.08	6.43	1.47	28.96	16.07	28.87	18.27
15	3.56	0.85	4.00	1.13	27.07	15.87	28.47	18.18
20	3.09	0.78	3.67	1.10	25.64	15.82	28.07	17.36
25	2.66	0.83	3.23	1.00	24.47	**15.60**	27.47	17.00
30	2.48	**0.75**	2.67	**0.87**	24.09	15.87	26.80	17.18
35	2.51	0.80	2.67	1.10	23.67	15.78	26.51	17.11
40	2.33	0.83	2.53	1.10	23.64	15.84	26.36	17.00
45	2.28	0.78	2.37	1.00	23.67	16.22	26.24	**16.93**
50	**2.16**	**0.75**	**2.30**	1.07	**23.21**	16.11	**26.16**	17.20

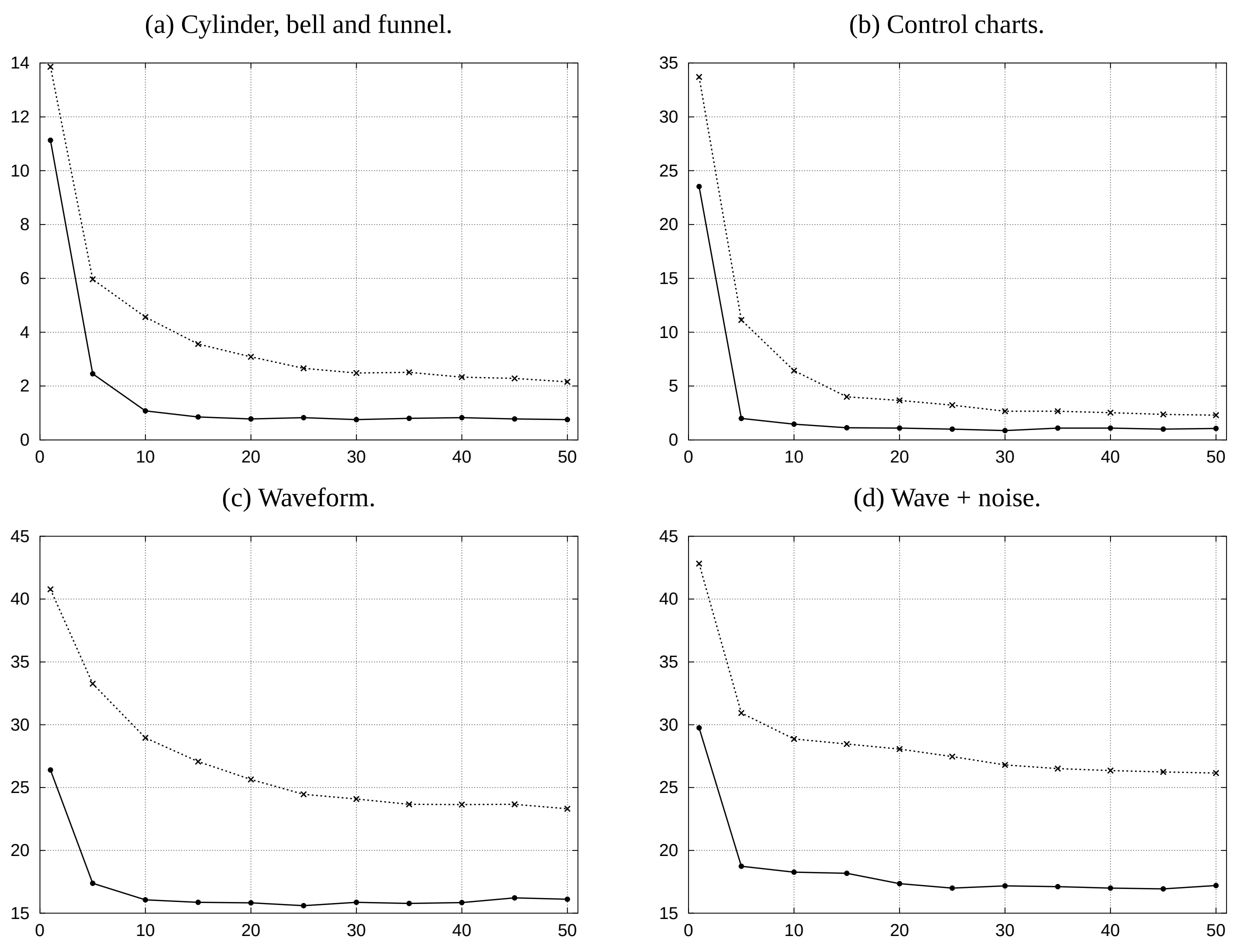

Fig. 2. Graphs of the results for the different data sets. For each one, the maximum and combined errors are plotted.

B.3 Waveform.

The best result in this case is obtained in the iteration 25, with an error of 15.60. The error of an optimal Bayes classifier on this data set, obtained analytically from the functions that generate the examples, is approximately 14 [12].

This data set is frequently used for testing classifiers. It has also been tested with boosting (and other methods of combining classifiers), over the raw data, in different works. Some of them:

- Applied to C4.5, with 10 iterations, 300 examples in total, 10 10-fold cross validations, obtaining a best error of 19.77 [14].
- Again, applied to C4.5, with 10 iterations, 10 10-fold cross validations, obtaining a best error of 19.13 [15].
- The error reported in [16] is 15.21, using AdaBoost.M1 and C4.5. The number of iterations considered is at most 100, but they stop the process if the error of the base classifier is greater than

0.5 or equal to 0.

- MultiBoosting (a combination of boosting and wagging) over C4.5, 300 examples in total, with a best result of 21.6 with 10 iterations and 19.3 with 100 iterations [17].
- It has also been tested in pruning boosting, but the concrete results for this data set are not available, only the global results obtained with a set of data sets and the relative results between different methods [18].

These results are obtained using base classifiers, trees, much more complex than out base classifiers (interval literals).

B.4 Wave + Noise.

Our best result in this case is an error of 16.93, in the iteration 45. Again, the error of an optimal Bayes classifier on this data set is of 14.
This data set was tested with bagging, boosting and variants over MC4 (similar to C4.5) [19], using 1000 examples for training and 4000 for test, and 25 iterations. Their results are in graphs, not in tables, so the exact value is not available, it appears that their best result is approximately 17.5.

V. Conclusions and Future Work

A time series classification system has been presented. It is based on boosting very simple classifiers. The individual classifiers are formed only by one literal. The predicates used are based on the presence of a variable in a region during an interval.
Experiments on several different data sets show that the proposed method is highly competitive with previous approaches. On several data sets, the proposed method achieves better than all previously reported results we are aware of. This is especially interesting if we consider that at the present stage:

- The individual classifiers are very simple.
- Only literals based on intervals are used.
- The method for handling multiclass problems is also very simple.

These facts suggest that there is enough room for future improvements.

Acknowledgements

To the maintainers of the ML [13] and KDD [11] UCI Repositories. To David Aha for donating the *Wave* data set and to Robert J. Alcock for donating the *Control Charts* data set.

References

[1] M. Kubat, I. Koprinska, and G. Pfurtscheller, “Learning to classify biomedical signals”, in *Machine Learning and Data Mining*, R.S. Michalski, I. Bratko, and M. Kubat, Eds., pp. 409–428. John Wiley & Sons, 1998.

[2] Carlos J. Alonso González and Juan J. Rodríguez Diez, “A graphical rule language for continuous dynamic systems”, in *Computational Intelligence for Modelling, Control and Automation. Evolutionary Computation & Fuzzy Logic for Intelligent Control, Knowledge Acquisition & Information Retrieval*, Masoud Mohammadian, Ed., Amsterdam, Netherlands, 1999, vol. 55 of *Concurrent Systems Engineering Series*, pp. 482–487, IOS Press.

[3] D.J. Berndt and J. Clifford, “Finding patterns in time series: a dynamic programming approach”, in *Advances in Knowledge Discovery and Data Mining*, U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds., pp. 229–248. AAAI Press / MIT Press, 1996.

[4] Mohammed Waleed Kadous, “Learning comprehensible descriptions of multivariate time series”, in *16th International Conference of Machine Learning (ICML-99)*, Ivan Bratko and Saso Dzeroski, Eds. 1999, Morgan Kaufmann.

[5] Robert E. Schapire, “A brief introduction to boosting”, in *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, Thomas Dean, Ed. 1999, pp. 1401–1406, Morgan Kaufmann.

[6] Juan J. Rodríguez Diez and Carlos J. Alonso González, “Temporal series classification through clauses restricted to literals on intervals”, in *8th Conference of the Spanish Association for Artificial Intelligence (CAEPIA’99)*, Ana M. García Serrano, Ramón Rizo, Serafín Moral, and Francisco Toledo, Eds., Murcia, Spain, November 1999, pp. 125–132.

[7] Y. Freund and R. Schapire, “Experiments with a new boosting algorithm”, in *13th International Conference om Machine Learning (ICML-96)*, Bari, Italy, 1996, pp. 148–156.

[8] Robert E. Schapire and Yoram Singer, “Improved boosting algorithms using confidence-rated predictions”, in *11th Annual Conference on Computational Learning Theory (COLT-98)*. 1998, pp. 80–91, ACM.

[9] Naoki Saito, *Local Feature Extraction and Its Applications Using a Library of Bases*, PhD thesis, Department of Mathematics, Yale University, 1994.

[10] Robert J. Alcock and Yannis Manolopoulos, “Time-series similarity queries employing a feature-based approach”, in *7th Hellenic Conference on Informatics*, Ioannina, Greece, 1999.

[11] Stephen D. Bay, “The UCI KDD archive”, 1999, `http://kdd.ics.uci.edu/`.

[12] L. Breiman, J. H. Friedman, A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Chapman & Hall, New York, 1993.

[13] C.L. Blake and C.J. Merz, “UCI repository of machine learning databases”, 1998, `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[14] J. Ross Quinlan, “Boosting, bagging, and C4.5”, in *13th National Conference on Artificial Inteligence (AAAI’96)*. 1996, pp. 725–730, AAAI Press.

[15] Michael Harries, “Boosting a strong learner: Evidence against the minimum margin”, in *16th International Conference of Machine Learning (ICML-99)*, Ivan Bratko and Saso Dzeroski, Eds. 1999, Morgan Kaufmann.

[16] Thomas G. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization”, *Machine Learning*, 1999.

[17] Geoffrey I. Webb, “MultiBoosting: A technique for combining boosting and wagging”, *Machine learning*, 1999.

[18] Dragos D. Margineantu and Thomas G. Dietterich, “Pruning adaptive boosting”, in *14th International Conference on Machine Learning (ICML-97)*, San Francisco, 1997, pp. 211–218, Morgan Kaufmann.

[19] Eric Bauer and Ron Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting and variants”, *Machine Learning*, vol. 36, no. 1/2, pp. 105–139, 1999.